THE UNIVERSITY OF CHICAGO

Overview of classical ML: classification methods and decision trees

Decision Trees

Cong Ma

Outline

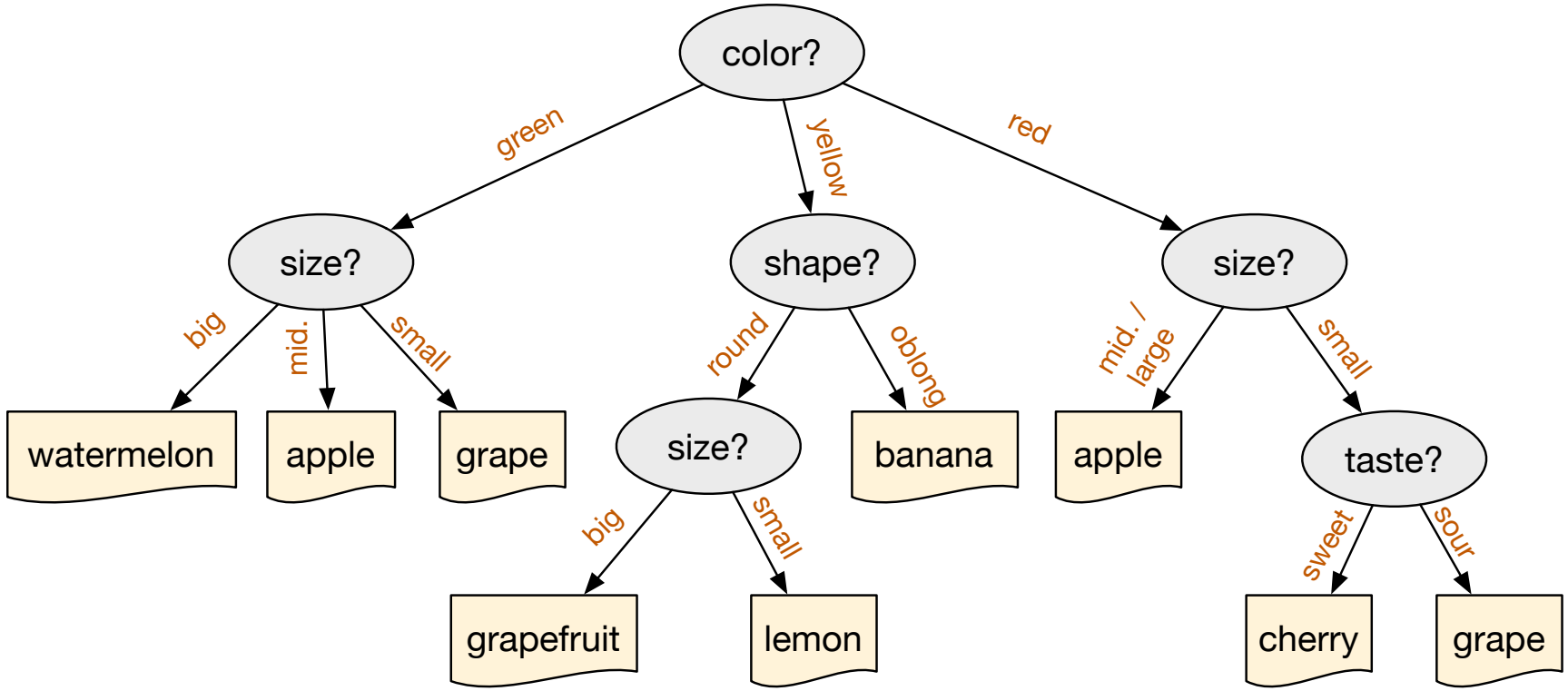
- What's a decision tree?
- Regression tree:
  - How to grow a tree: decrease in squared error
  - How to prune a tree
  - How to predict given a tree
- Classification tree
  - How to grow a tree: misclassification rate, information gain, Gini index
  - How to predict
- Summary

Tree based methods

- Divide the input space into a number of simple regions
- Use simple prediction rules in each region

Adaptive feature selection

- Prediction based on (a sequence of) decision rules

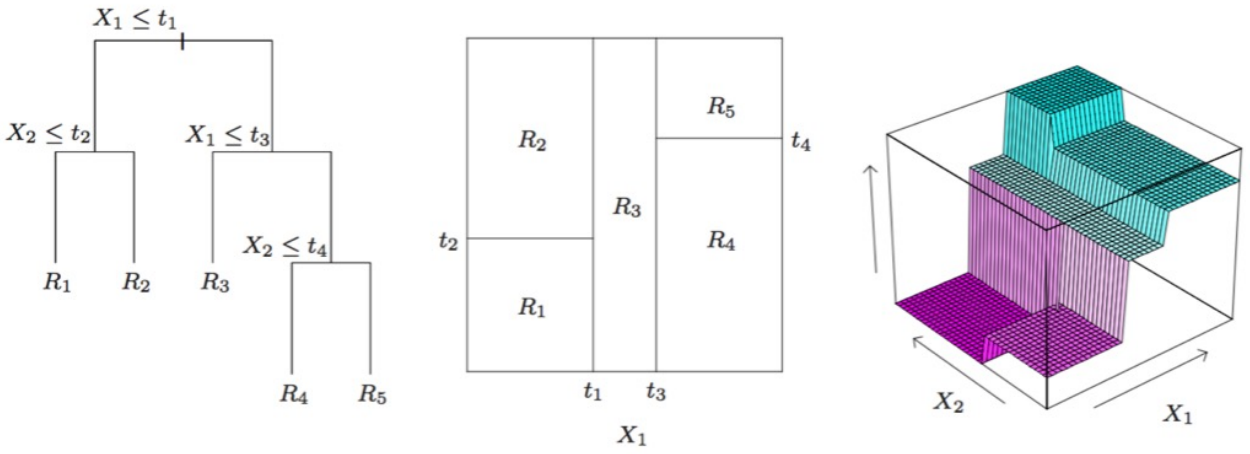


Regression trees

Trees

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m).$$

Build a binary tree, splitting along axes



Goal

- The goal is to find boxes $R_1, \ldots, R_J$ that minimize the RSS, given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box.

More details of the tree-building process

- Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes.
- For this reason, we take a *top-down*, *greedy* approach that is known as recursive binary splitting.
- The approach is *top-down* because it begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- It is *greedy* because at each step of the tree-building process, the *best* split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

How to grow a regression tree

$$R_1(j,s) = \{X | X_j \leq s\} \text{ and } R_2(j,s) = \{X | X_j > s\}.$$

Then we seek the splitting variable j and split point s that solve

$$\min_{j,\, s} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right].$$

Pruning a tree

- The process described above may produce good predictions on the training set, but is likely to *overfit* the data, leading to poor test set performance. *Why?*
- A smaller tree with fewer splits (that is, fewer regions $R_1, \ldots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias.
- One possible alternative to the process described above is to grow the tree only so long as the decrease in the RSS due to each split exceeds some (high) threshold.
- This strategy will result in smaller trees, but is too *short-sighted:* a seemingly worthless split early on in the tree might be followed by a very good split — that is, a split that leads to a large reduction in RSS later on.

Tree pruning

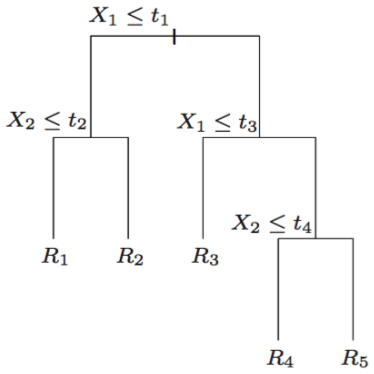
- Greedily grow the tree is prone to overfitting
- Tree pruning phase
  - Searching over all trees $\mathcal{T}$ and find the one with the best fit to data and smallest size

$$\min_{\mathcal{T}} -\sum_{v \in \mathcal{T}} L(S_v) + \lambda |\mathcal{T}|$$

- We can prune back tree branches (i.e. merge a pair of leaf nodes) recursively to choose the tree that minimizes the above objective
  - Due to the greedy nature for the growth phase, the combined growth + pruning process is not guaranteed to find the optimal tree

Learning decision trees

- Start from empty decision tree
- Split on next best attribute (feature)
  - Use, for example, information gain to select attribute
  - Split on $\arg\max_i IG(X_i) = \arg\max_i H(Y) - H(Y \mid X_i)$
- Recurse
- Prune



$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m).$$

Classification tree

- How to split a node?
- How to predict in the end?

Another Measure: Gini Index

- Gini index: Used in CART, and also in IBM IntelligentMiner
- If a data set D contains examples from n classes, gini index, $gini(D)$ is defined as
  - $gini(D) = 1 - \sum_{j=1}^{n} p_j^2$
    - p_j is the relative frequency of class j in D
- If a data set D is split on A into two subsets D_1 and D_2, the $gini$ index $gini(D)$ is defined as
  - $gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$
- Reduction in Impurity:
  - $\Delta gini(A) = gini(D) - gini_A(D)$
- The attribute provides the smallest $gini_{split}(D)$ (or the largest reduction in impurity) is chosen to split the node (*need to enumerate all the possible splitting points for each attribute*)

16

Computation of Gini Index

- Example: D has 9 tuples in buys_computer = "yes" and 5 in "no"

$$gini(D) = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 = 0.459$$

- Suppose the attribute income partitions D into 10 in D_1: {low, medium} and 4 in D_2
  - $gini_{income \in \{low,medium\}}(D) = \frac{10}{14} gini(D_1) + \frac{4}{14} gini(D_2)$

$$= \frac{10}{14}\left(1 - \left(\frac{7}{10}\right)^2 - \left(\frac{3}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right) = 0.443$$

$$= Gini_{income \in \{high\}}(D)$$

  - $Gini_{\{low,high\}}$ is 0.458; $Gini_{\{medium,high\}}$ is 0.450
  - Thus, split on the {low,medium} (and {high}) since it has the lowest Gini index
- All attributes are assumed continuous-valued
- May need other tools, e.g., clustering, to get the possible split values
- Can be modified for categorical attributes

17

THE UNIVERSITY OF CHICAGO

STAT 37710 / CMSC 35400 / CAAM 37710 Machine Learning

Bagging & Random Forests

Cong Ma

Recall: decision trees

- Decision Trees are
  - low bias, high variance models
    - Unless you regularize a lot…
    - …but then often worse than Linear Models
  - highly non-linear
    - Can easily overfit
    - Different training samples can lead to very different trees

$$\underbrace{\mathbb{E}_D\left[\left(y-\hat{h}_D(\mathbf{x})\right)^2\right]}_{\text{expected error}} = \underbrace{\mathbb{E}_D\left[\hat{h}_D(\mathbf{x})-y\right]^2}_{\text{bias}} + \underbrace{\mathbb{E}_D\left[\left(\hat{h}_D(\mathbf{x})-\mathbb{E}_{D'}\hat{h}_{D'}(\mathbf{x})\right)^2\right]}_{\text{variance}}$$

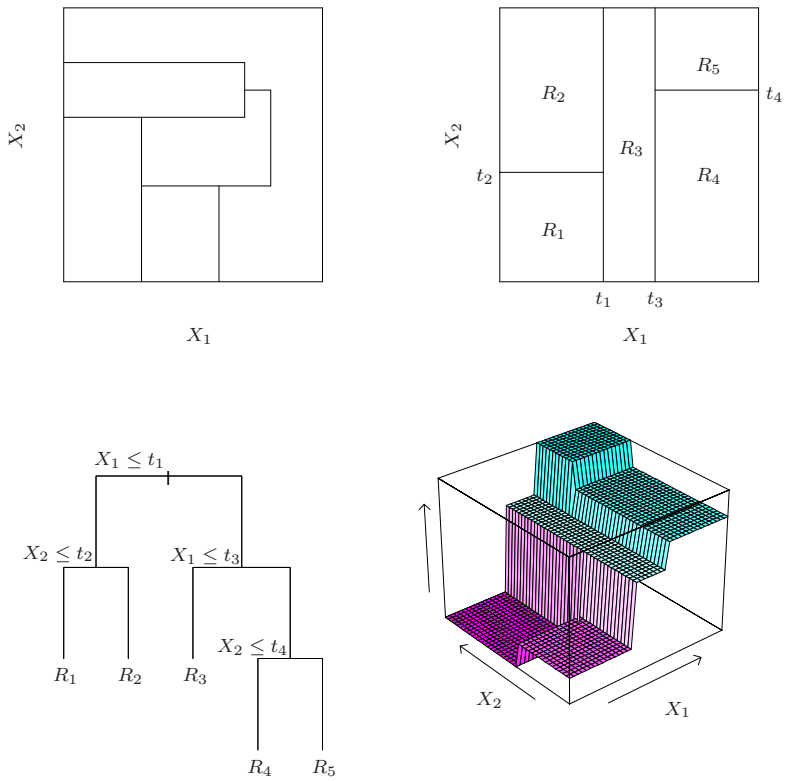


FIGURE 9.2. *Partitions and CART. Top right panel shows a partition of a two-dimensional feature space by recursive binary splitting, as used in CART, applied to some fake data. Top left panel shows a general partition that cannot be obtained from recursive binary splitting. Bottom left panel shows the tree corresponding to the partition in the top right panel, and a perspective plot of the prediction surface appears in the bottom right panel.*

How to improve decision trees?

- What's the problem of decision tree?
  - Low bias but high variance
- We'd like to keep the low bias, but decrease the variance
  - Key idea: build multiple trees and take the average
  - We know averaging reduces variance (Caveat!)

Average over multiple different datasets

- Goal: reduces variance
- Ideal setting:
  - many training sets D’
    - sample independently
  - train model using each D’
  - average predictions

P(x,y)

Person	Age	Male?	Height > 55”
James	11	1	1
Jessica	14	0	1
Alice	14	0	1
Amy	12	0	1
Bob	10	1	1
Xavier	9	1	0
Cathy	9	0	1
Carol	13	0	1
Eugene	13	1	0
Rafael	12	1	1
Dave	8	1	0
Peter	9	1	0
Henry	13	1	0
Erin	11	0	0
Rose	7	0	0
Iain	8	1	1
Paulo	12	1	0
Frank	9	1	1
Jill	13	0	0
Leon	10	1	0
Sarah	12	0	0
Gena	8	0	0
Patrick	5	1	1

D’

Person	Age	Male?	Height > 55”
Alice	14	0	1
Bob	10	1	1
Carol	13	0	1
Dave	8	1	0
Erin	11	0	0
Frank	9	1	1
Gena	8	0	0

“Bagging Predictors” [Leo Breiman, 1994]

http://statistics.berkeley.edu/sites/default/files/tech-reports/421.pdf

Bagging

- Goal: reduces variance
- In practice:
  - fixed training set D
    - Resample D' with replacement from D
  - train model using each D'
  - average predictions

D

Person	Age	Male?	Height > 55"
James	11	1	1
Jessica	14	0	1
Alice	14	0	1
Amy	12	0	1
Bob	10	1	1
Xavier	9	1	0
Cathy	9	0	1
Carol	13	0	1
Eugene	13	1	0
Rafael	12	1	1
Dave	8	1	0
Peter	9	1	0
Henry	13	1	0
Erin	11	0	0
Rose	7	0	0
Iain	8	1	1
Paulo	12	1	0
Frank	9	1	1
Jill	13	0	0
Leon	10	1	0
Sarah	12	0	0
Gena	8	0	0
Patrick	5	1	1

D'

Person	Age	Male?	Height > 55"
Alice	14	0	1
Bob	10	1	1
Carol	13	0	1
Dave	8	1	0
Erin	11	0	0
Frank	9	1	1
Gena	8	0	0

"Bagging Predictors" [Leo Breiman, 1994]

http://statistics.berkeley.edu/sites/default/files/tech-reports/421.pdf

Bagging = Bootstrap Aggregating

- Learns a predictor by aggregating the predictors learned over multiple random draws (bootstrap samples) from the training data
  - A bootstrap sample of size m from $D : \{(\mathbf{x}_i, y_i), i = 1, \ldots, n\}$ is

$$\{(\mathbf{x}'_i, y'_i), i = 1, \ldots, m\}$$

where each (x_i', y_i') is drawn uniformly at random from D (with replacement)

Bagged trees

Algorithm:

1. Obtain *B* bootstrap resamples of our training sample
2. For each resample, grow a large (low bias, high variance) tree
3. Average/aggregate predictions from all of the trees
   a. Regression: take the mean of the *B* predictions
   b. Classification: take the majority vote of the *B* predictions

Aggregating weak predictors

- Imagine we have a model we can fit to the training data to produce a predictor that we use to predict $E(Y|X=x)$
  - E.g. a decision tree or logistic regression

- With bagging, we
  - compute B different bootstrap samples
  - learn a predictor for each one
  - aggregate the predictors to form the target predictor

Bootstrap

- Assume you have a sample $X_1, ..., X_n$ of points and, say, an estimate $\hat{\Theta}$ of a true parameter Θ of this population. You would like to know the distribution of the estimate $\hat{\Theta}$ (for example, because you want to construct confidence sets).
- You now draw a subsample of m points of the original sample (with our without replacement), and on this subsample you compute an estimate of the parameter you are interested in.
- You repeat this procedure B times, resulting in B bootstrap estimates $\hat{\Theta}_1, ..., \hat{\Theta}_B$.
- This set now gives an "indication" about how your estimate is distributed, and you can compute its mean, its variance, confidence sets, etc.

Bagging

- As in bootstrap, you generate B bootstrap samples of your original sample, and on each of them compute the estimate you are interested in: $\hat{\Theta}_1, ..., \hat{\Theta}_B$
- As your final estimate, you then take the average: $\hat{\Theta}_{bag} = mean(\hat{\Theta}_1, ..., \hat{\Theta}_B)$.
- The advantage of this procedure is that the estimate Θ_{bag} can have a much smaller variance than each of the individual estimates $\hat{\Theta}_b$:
  - If the estimates $\hat{\Theta}_b$ were i.i.d. with variance σ^2, then the variance of $\hat{\Theta}_{bag}$ would be σ^2/B .
  - If the estimates are identically distributed but have a (hopefully small) positive pairwise correlation ρ, then the variance of $\hat{\Theta}_{bag}$ is $\rho\sigma^2 + (1-\rho)\frac{\sigma^2}{B}$. If ρ is small and B is large, this is good.

Decorrelate the trees

- Key: we'd like "diversity" in the trees we build, or further decorrelate the trees we build
- Use random features in splitting the nodes!

Random Forests

- Goal: reduce variance
  - Bagging can only do so much
  - Resampling training data

- Random Forests: sample data & features!
  - Sample S’
  - Train DT
    - At each node, sample features
  - Average predictions

Random Forests

- Extension of bagging to sampling features

- Generate bootstrap D’ from D
  - Train DT top-down on D’
  - Each node, sample subset of features for splitting
    - Can also sample a subset of splits as well

- Average predictions of all DTs

“Random Forests – Random Features” [Leo Breiman, 1997]
http://oz.berkeley.edu/~breiman/random-forests.pdf

Algorithm for random forest

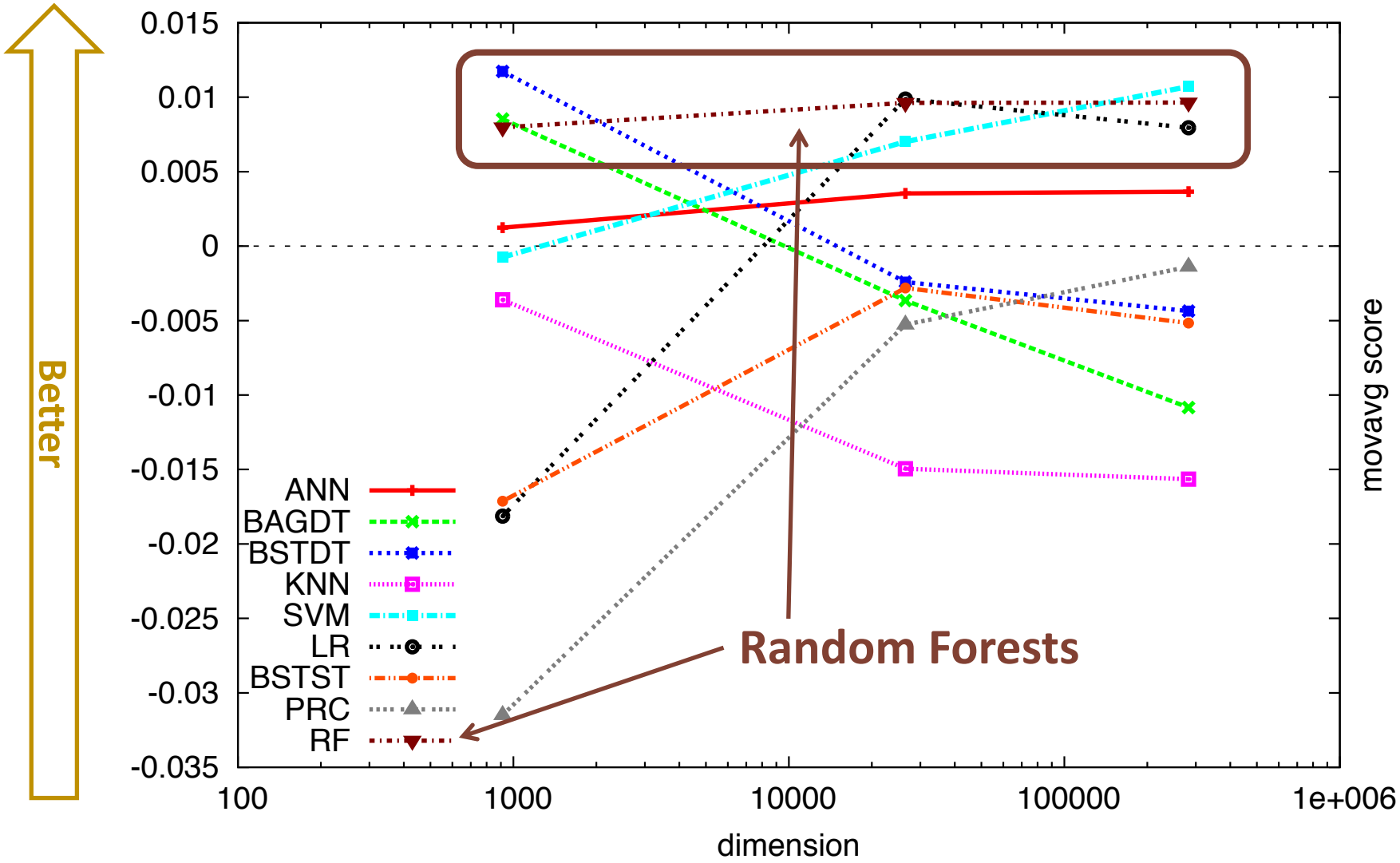
Algorithm 15.1 *Random Forest for Regression or Classification.*

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote } \{\hat{C}_b(x)\}_1^B$.



Average performance over many datasets
Random Forests perform the best

"An Empirical Evaluation of Supervised Learning in High Dimensions"
Caruana, Karampatziakis & Yessenalina, ICML 2008

THE UNIVERSITY OF CHICAGO

STAT 37710 / CMSC 35400 / CAAM 37710
Machine Learning

Boosting

Cong Ma

AdaBoost for binary classification

We begin by describing the most popular boosting algorithm due to Freund and Schapire (1997) called "AdaBoost.M1." Consider a two-class problem, with the output variable coded as $Y \in \{-1, 1\}$. Given a vector of predictor variables X, a classifier $G(X)$ produces a prediction taking one of the two values $\{-1, 1\}$. The error rate on the training sample is

$$\overline{\text{err}} = \frac{1}{N} \sum_{i=1}^{N} I(y_i \neq G(x_i)),$$

and the expected error rate on future predictions is $\mathrm{E}_{XY} I(Y \neq G(X))$.

- Purpose of Boosting: sequentially apply the weak classification algorithm to repeatedly modified versions of the data, thereby producing a sequence of weak classifiers

Weak learner to strong learner?

- 1988 Kearns and Valiant: "Can **weak learners** be combined to create a **strong learner?**"

Weak learner definition (informal):

An algorithm $\mathcal{A}$ is a *weak learner* for a hypothesis class $\mathcal{H}$ that maps $\mathcal{X}$ to $\{-1, 1\}$ if for all input distributions over $\mathcal{X}$ and $h \in \mathcal{H}$, we have that $\mathcal{A}$ correctly classifies h with error at most $1/2 - \gamma$

- 1990 Robert Schapire: "Yup!"
- 1995 Schapire and Freund: "Practical for 0/1 loss" AdaBoost
- 2001 Friedman: "Practical for arbitrary losses"

Figure for AdaBoost

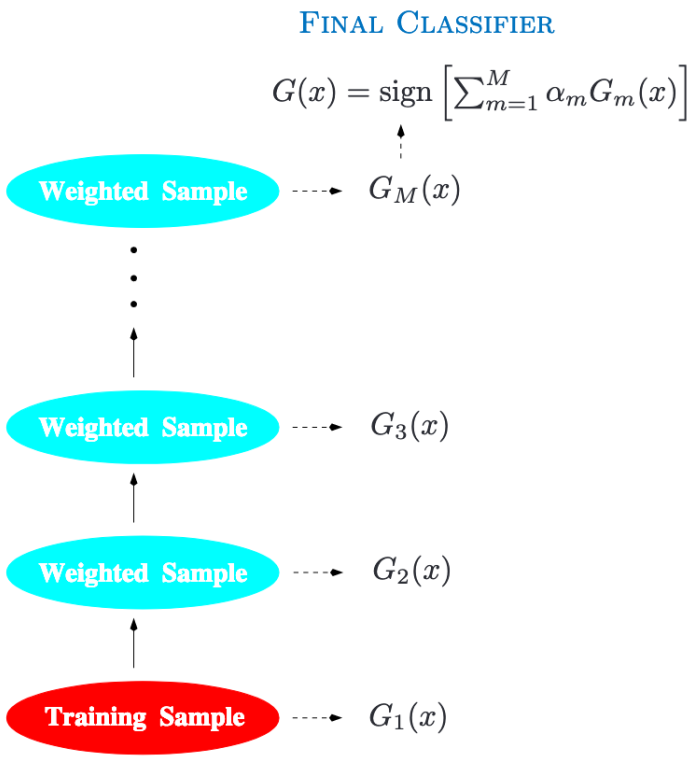


FIGURE 10.1. *Schematic of AdaBoost. Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction.*

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathscr{X}$, $y_i \in \{-1, +1\}$.

Initialize $\boxed{D_1(i) = 1/m \text{ for } i = 1, \ldots, m.}$ ← Initial Distribution of Data

For $t = 1, \ldots, T$:

- $\boxed{\begin{array}{l}\text{Train weak learner using distribution } D_t. \\ \text{Get weak hypothesis } h_t : \mathscr{X} \to \{-1, +1\}.\end{array}}$ ← Train model
- Aim: select h_t with low weighted error:

$$\boxed{\varepsilon_t = \Pr_{i \sim D_t}\left[h_t(x_i) \neq y_i\right].}$$

← Error of model

- Choose $\boxed{\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \varepsilon_t}{\varepsilon_t}\right).}$ ← Coefficient of model
- Update, for $i = 1, \ldots, m$:

$$\boxed{D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}}$$

← Update Distribution

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$\boxed{H(x) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).}$$

← Final average

Theorem: training error drops exponentially fast

http://rob.schapire.net/papers/explaining-adaboost.pdf

Boosting fits an additive model

The success of boosting is really not very mysterious. The key lies in expression (10.1). Boosting is a way of fitting an additive expansion in a set of elementary "basis" functions. Here the basis functions are the individual classifiers $G_m(x) \in \{-1, 1\}$. More generally, basis function expansions take the form

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m), \tag{10.3}$$

where $\beta_m, m = 1, 2, \ldots, M$ are the expansion coefficients, and $b(x; \gamma) \in \mathbb{R}$ are usually simple functions of the multivariate argument x, characterized by a set of parameters γ. We discuss basis expansions in some detail in Chapter 5.

Typically these models are fit by minimizing a loss function averaged over the training data, such as the squared-error or a likelihood-based loss function,

$$\min_{\{\beta_m,\gamma_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i;\gamma_m)\right). \tag{10.4}$$

Algorithm 10.2 *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:
   (a) Compute
   $$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i;\gamma)).$$
   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x;\gamma_m)$.

Boosting for regression

$$L(y, f(x)) = (y - f(x))^2,$$

one has

$$\begin{aligned} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) &= (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2 \\ &= (r_{im} - \beta b(x_i; \gamma))^2, \end{aligned} \tag{10.7}$$

where $r_{im} = y_i - f_{m-1}(x_i)$ is simply the residual of the current model

AdaBoost with exponential loss

We now show that AdaBoost.M1 (Algorithm 10.1) is equivalent to forward stagewise additive modeling (Algorithm 10.2) using the loss function

$$L(y, f(x)) = \exp(-y\, f(x)). \tag{10.8}$$

For AdaBoost the basis functions are the individual classifiers $G_m(x) \in \{-1, 1\}$. Using the exponential loss function, one must solve

$$(\beta_m, G_m) = \arg\min_{\beta, G} \sum_{i=1}^{N} \exp[-y_i(f_{m-1}(x_i) + \beta\, G(x_i))]$$

for the classifier G_m and corresponding coefficient β_m to be added at each step. This can be expressed as

$$(\beta_m, G_m) = \arg\min_{\beta, G} \sum_{i=1}^{N} w_i^{(m)} \exp(-\beta\, y_i\, G(x_i)) \tag{10.9}$$

with $w_i^{(m)} = \exp(-y_i\, f_{m-1}(x_i))$. Since each $w_i^{(m)}$ depends neither on β

Why does boosting work?

- AdaBoost can be understood as a procedure for greedily minimizing the exponential loss over T rounds:

$$\ell(y_i, h(\mathbf{x}_i)) = \exp(-y_i h(\mathbf{x}_i)) \qquad \text{where} \qquad h(\mathbf{x}_i) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}_i)$$

  - Why?

"Prediction games and arcing classifiers" [Leo Breiman, 1999]
https://www.stat.berkeley.edu/~breiman/games.pdf

Interpretation of Adaboost

- Choosing the first classifier

$$(\alpha_1, \hat{h}_1) = \arg\min_{\alpha,h} \sum_{i=1}^{m} \ell(y_i, \alpha h(\mathbf{x}_i)) = \arg\min_{\alpha,h} \sum_{i=1}^{m} \exp(-y_i \cdot \alpha h(\mathbf{x}_i))$$

- Update at round t

$$\tilde{h}_{t-1}(\mathbf{x}) = \sum_{\tau=1}^{t-1} \alpha_\tau \hat{h}_\tau(\mathbf{x})$$

$$(\alpha_t, \hat{h}_t) = \arg\min_{\alpha,h} \sum_{i=1}^{m} \ell(y_i, \tilde{h}_{t-1}(\mathbf{x}) + \alpha h(\mathbf{x}_i))$$

$$= \arg\min_{\alpha,h} \sum_{i=1}^{m} \exp(-y_i \cdot (\tilde{h}_{t-1}(\mathbf{x}) + \alpha h(\mathbf{x}_i)))$$

Interpretation of Adaboost

$$(\alpha_t, \hat{h}_t) = \arg\min_{\alpha,h} \sum_{i=1}^{m} \exp(-y_i \cdot (\tilde{h}_{t-1}(\mathbf{x}_i) + \alpha h(\mathbf{x}_i)))$$

$$= \arg\min_{\alpha,h} \sum_{i=1}^{m} \underbrace{\exp(-y_i \tilde{h}_{t-1}(\mathbf{x}_i))}_{w_i^{(t)}} \exp(-y_i \cdot \alpha h(\mathbf{x}_i))$$

- Correcting the label for misclassified points
  - Giving those points higher weights when training classifier in future iterations

- We will solve h and α separately

Solving for h

- Fix α, $\hat{h}_t = \arg\min_h \sum_{i=1}^{m} w_i^{(t)} \exp(-y_i \cdot \alpha h(\mathbf{x}_i))$

Solving for α

- Now solve for α

AdaBoost weight update

- Putting things together,

$$\hat{h}_t = \arg\min_h \underbrace{\frac{1}{\sum_{i=1}^m w_i^{(t)}} \sum_{i=1}^m w_i^{(t)} \mathbb{1}[h(\mathbf{x}_i) \neq y_i]}_{\mathrm{err}_{\hat{h}_t}} \qquad \alpha_t = \frac{1}{2} \ln \left(\frac{1 - \mathrm{err}_{\hat{h}_t}}{\mathrm{err}_{\hat{h}_t}} \right)$$

- Therefore, weights for next round are

$$\begin{aligned} w_i^{(t+1)} &= \exp(-y_i(\tilde{h}_{t-1}(\mathbf{x}) + \alpha_t \hat{h}_t(\mathbf{x})) \\ &= \underbrace{\exp(-y_i \tilde{h}_{t-1}(\mathbf{x}_i))}_{w_i^{(t)}} \cdot \exp(-\alpha_t y_i \hat{h}_t(\mathbf{x}_i)) \end{aligned}$$

Why do we care about exponential loss?

- Fisher consistent loss

It is easy to show (Friedman et al., 2000) that

$$f^*(x) = \arg\min_{f(x)} \mathrm{E}_{Y|x}(e^{-Yf(x)}) = \frac{1}{2}\log\frac{\Pr(Y=1|x)}{\Pr(Y=-1|x)}, \qquad (10.16)$$

Gradient boosting

- Consider a generic loss function
  - E.g. squared loss, exponential loss

- Given current predictor $\tilde{h}_{t-1}(\mathbf{x})$, we aim to find new predictor h(x) so that the sum $\tilde{h}_{t-1}(\mathbf{x}) + h(\mathbf{x})$ pushes the loss towards its minimum as quickly as possible

- Gradient boosting: choose h in the direction of the negative gradient of the loss

Gradient boosting

- Fit a model to the negative gradients
- XGBoost is a python package for "extreme" gradient boosting
  - Folk wisdom: knowing logistic regression and XGBoost gets you 95% of the way to a winning Kaggle submission for most competitions
  - State-of-the-art prediction performance
    - Won Netflix Challenge
    - Won numerous KDD Cups
    - Industry standard

Gradient Boosting

start with an initial model, e.g. $\tilde{h}_1(x) = \frac{1}{n}\sum_{i=1}^{n} y_i$

for $b = 1, 2, \ldots$

calculate negative gradients

$$-g(x_i) = -\frac{\partial L(y_i, \tilde{h}_b(x_i))}{\partial h_b(x_i)}$$

fit a model h_b (e.g. tree) to negative gradients: $h_b = \underset{h}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} L(-g(x_i), h(x_i))$

$$\tilde{h}_{b+1}(x) = \tilde{h}_b(x) + \beta_b h_b(x)$$

where β_b is a step size parameter we find computationally to minimize the loss.

if $\tilde{h}_{b+1} \approx \tilde{h}_b$, STOP

Willett & Chen (2020). "CMSC 35400: Machine Learning"

References & acknowledgement

- Hastie et al. (2009). “The Elements of Statistical Learning”
  - Ch 10.1 , “Boosting Methods”
  - Ch 10.4, “Exponential Loss and AdaBoost”
- Willett & Chen (2020). “CMSC 35400: Machine Learning”
- Yue (2018). “Machine Learning & Data Mining”
  - Lecture 5, “Decision Trees, Bagging & Random Forests”
- Schapire (2013). “Explaining AdaBoost”
  - http://rob.schapire.net/papers/explaining-adaboost.pdf